



Pontifícia Universidade Católica de Campinas
Faculdade de Engenharia de Computação -
FECOMP

Redes de Computadores A – Projeto
Chat

Carlos Henrique Vieira Marques RA: 18720367

Fabício Silva Cardoso RA: 18023481

Sumário

1. Introdução	3
2. Preparando o ambiente de desenvolvimento.....	4
3. Interações e Protocolos escolhidos	4
4. Formato do sistema de troca de mensagens.....	5
5. Análise dos Resultados.....	8
6. Conclusão	8

1. Introdução

Neste projeto o grupo procurou explorar e entender mais a fundo as diretrizes do protocolo TCP e a comunicação entre cliente através de um servidor. Utilizando os materiais de apoio disponibilizados pelo professor, pudemos desenvolver e conhecer mais a respeito da estrutura e funcionamento dos sistemas de redes. O objetivo dessa atividade é implementar um serviço de chat em linguagem C utilizando sockets.

O funcionamento do chat se dá por um servidor central que tem seu IP e portas fixados em 127.0.0.1 e 1234, respectivamente, sendo ele, responsável por receber os clientes e suas solicitações. O servidor não permite uma conexão direta entre os clientes, sempre fazendo assim o controle e filtro necessário antes que as mensagens sejam enviadas para seus respectivos destinos.

Por sua vez, os clientes ao se conectarem ao servidor, são cadastrados no sistema e assim serão reconhecidos pelos seus respectivos **usernames**, onde por esses nomes serão identificados caso seja solicitado em algum momento. Ao se desconectar do servidor, os clientes tem seu status alterado para offline e assim são excluídos da lista de clientes online.

2. Ambiente utilizado

O ambiente utilizado para desenvolvimento do projeto é composto pela IDE Visual Studio Code (VS Code) com algumas extensões, além de um terminal Linux adaptado para Windows, também conhecido como WSL. As ferramentas utilizadas foram escolhidas pois todo o grupo já estava habituado com sua utilização.

3. Preparando o ambiente

Para utilizar a IDE escolhida, foi necessário instalar duas extensões essenciais para o bom funcionamento, sendo elas a C/C++ e o Remote – WSL, além da instalação do próprio WSL, que pode ser feita a partir do Microsoft Store.

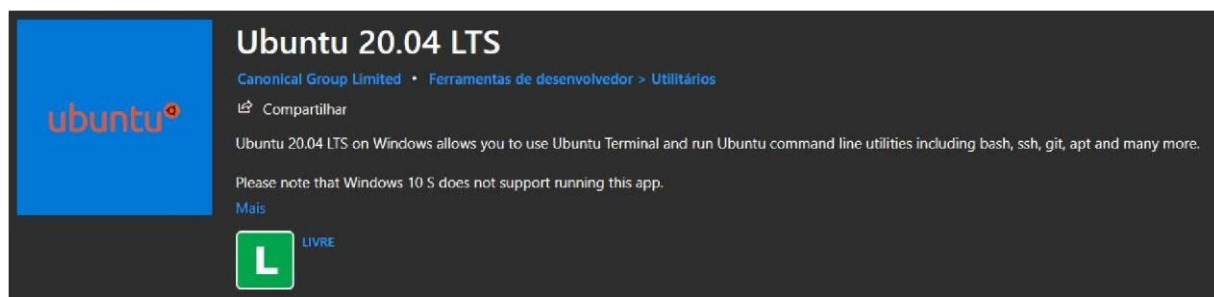


Figura 1 – WSL que foi utilizado durante o projeto

A extensão C/C++ foi instalada para permitir que o VS Code tenha suporte para a linguagem C, uma vez que ao baixa-lo, não existe suporte por padrão. Já a instalação do WSL se deve pela facilidade da utilização e menor trabalho para se utilizar recursos do sistema operacional que normalmente são bloqueados no Windows e dificultam o desenvolvimento desse modelo de aplicação, que são, em sua maioria, desenvolvidos nos sistemas Linux ou derivados. Além disso, foi necessário habilitar os recursos do Windows para utilização de uma extensão Linux no sistema operacional.

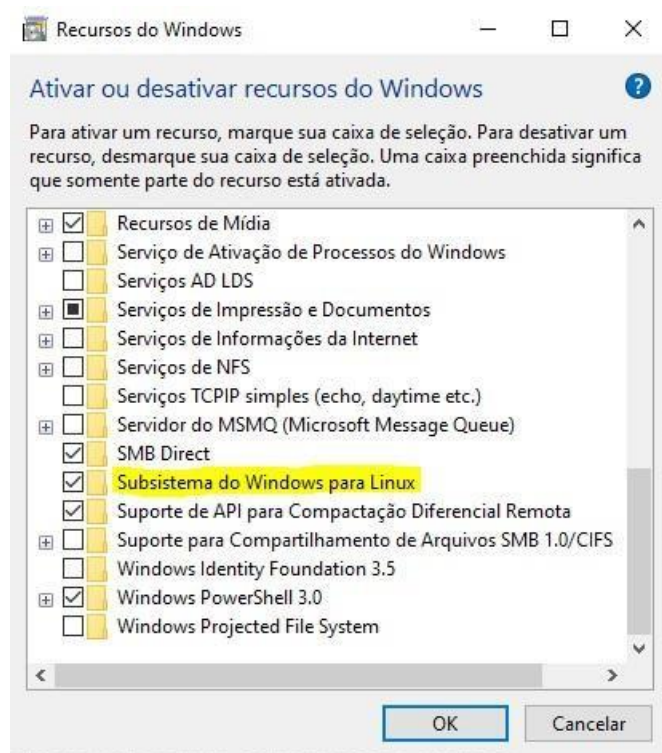


Figura 2 – Para o funcionamento do WSL é necessário habilitar essa opção no Windows

Após tudo instalado e configurado, foi dada a partida para o desenvolvimento da aplicação.

4. Interações e protocolos escolhidos

O projeto foi desenvolvido utilizando o protocolo TCP/IP, visando realizar as interações entre clientes através do servidor, permitindo assim um maior controle das mensagens enviadas.

O protocolo foi escolhido por conta de seu maior controle de mensagens, de erros e de pacotes enviados. O funcionamento pode ser visto no diagrama a seguir.

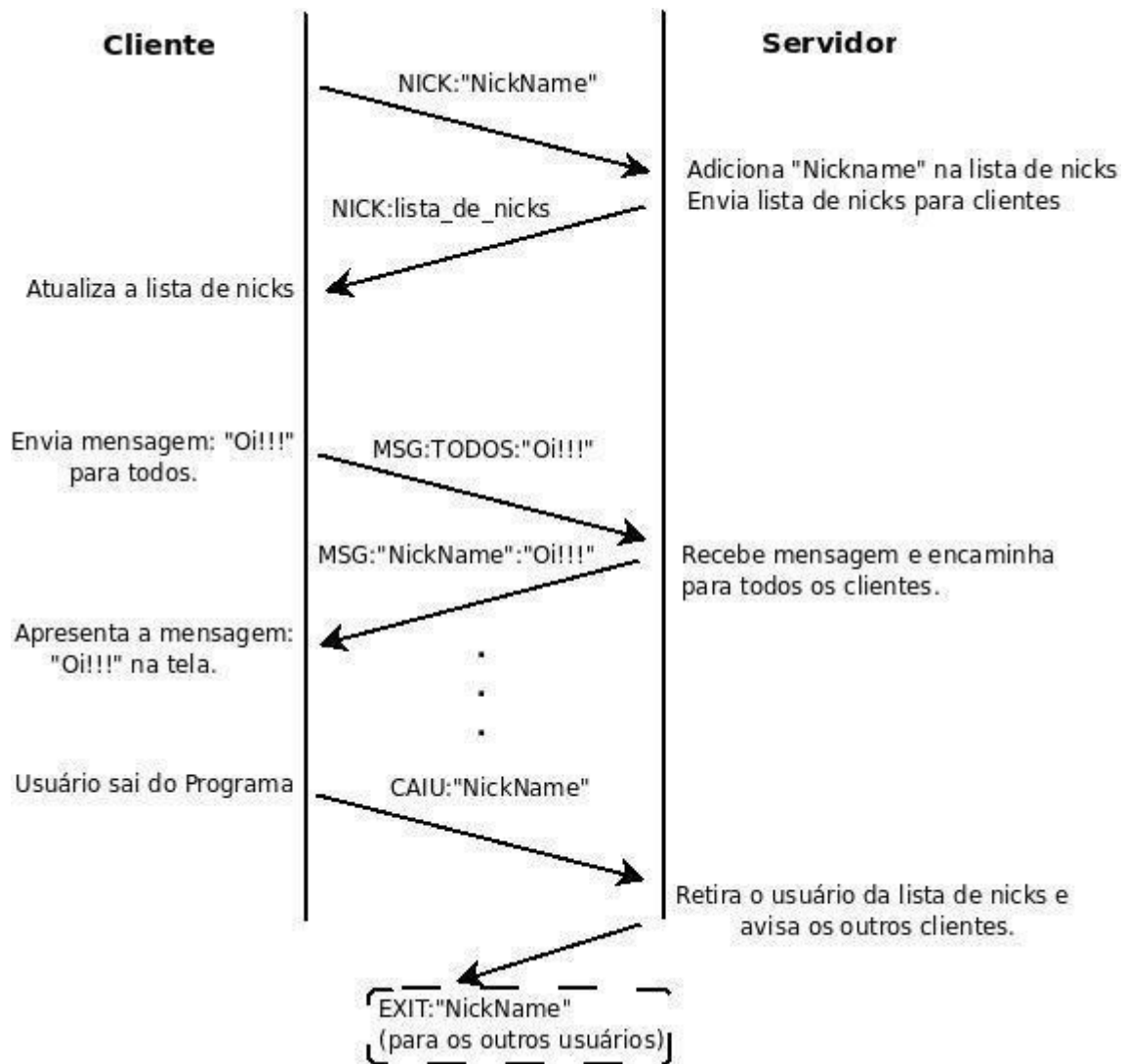


Figura 3 - Fluxo das informações do programa

4.Formato do sistema de troca de mensagens.

Como citado anteriormente, utilizamos o protocolo TCP como ponte para conexão de rede entre servidor e usuários. A seguir faremos uma breve descrição de como esse sistema de troca de mensagens funciona.

Antes de iniciar o sistema é necessário fazer uma atribuição de um socket, permitindo assim que o nosso servidor tenha uma porta e um IP atribuídos. Para o sistema todo funcionar, foi fixada a porta 1234 no servidor, permitindo assim que todos os clientes se conectem a uma mesma porta, não sendo necessário fazer um tratamento especial para cada cliente de forma individual.

Depois de criado o socket e fixada a porta, é necessário habilitar nosso servidor para receber mensagens e os mais diversos protocolos conhecidos. Isso é feito utilizando a chamada de sistema `bind()`, permitindo assim que seja feita a escolha do protocolo de comunicação.

Como o protocolo escolhido pelo grupo foi o TCP, é necessário fazer com que o servidor fique escutando na porta fixada por solicitações de clientes, onde isso é possível ser feito com o `listen()`. O `listen` permite escolhermos o número máximo de clientes que poderão se conectar ao mesmo tempo no servidor. Caso esse número seja excedido, quem solicitou é colocado em uma fila, onde fica aguardando algum cliente se desconectar para poder se conectar.

No projeto, o número máximo de clientes foi definido para 5, porém pode ser alterado para um número ilimitado de clientes. Quando a conexão é estabelecida de forma correta com o servidor, o usuário deve escolher por qual **username** ele será reconhecido no sistema. Após essa escolha, seu nome é adicionado na lista de usuários, caso seja sua primeira conexão, caso contrário apenas seu status é alterado na lista de clientes registrados.

Ao se conectar, o cliente terá um menu com diversas funcionalidades para ele aproveitar o sistema, sendo elas:

-quit: Sair do servidor.

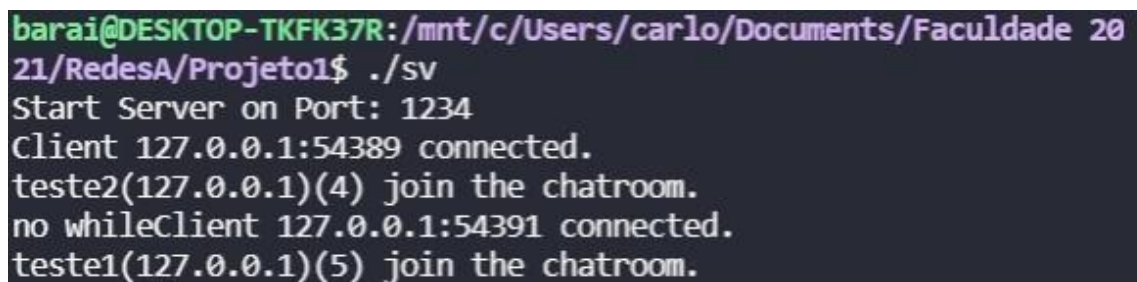
-all: Enviar para todos os usuários online.

-msg: Enviar uma mensagem privada.

-list: Mostrar todos os clientes e seu atual status -arq: enviar um arquivo para um cliente.

-help: solicitar ajuda sobre os comandos.

Segue abaixo algumas imagens de exemplo:



```
barai@DESKTOP-TKFK37R:/mnt/c/Users/carlo/Documents/Faculdade 20
21/RedesA/Projeto1$ ./sv
Start Server on Port: 1234
Client 127.0.0.1:54389 connected.
teste2(127.0.0.1)(4) join the chatroom.
no whileClient 127.0.0.1:54391 connected.
teste1(127.0.0.1)(5) join the chatroom.
```

Figura 4 – Servidor anunciado a chegada de novos Usuários.

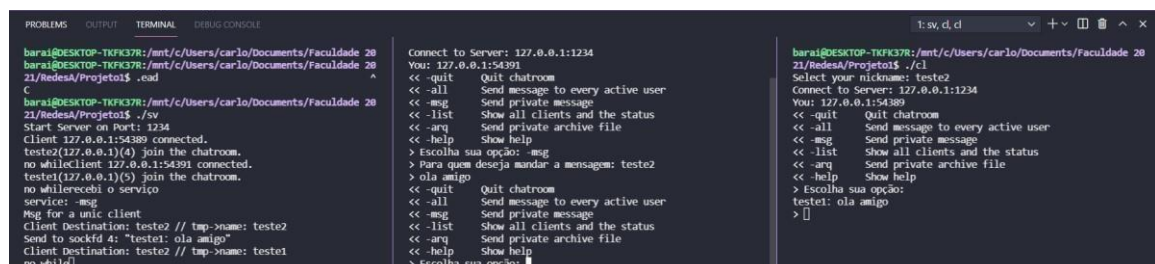
```
barai@DESKTOP-TKFK37R:/mnt/c/Users/carlo/Documents/Faculdade 20
21/RedesA/Projeto1$ ./cl
Select your nickname: teste1
Connect to Server: 127.0.0.1:1234
You: 127.0.0.1:54391
<< -quit      Quit chatroom
<< -all        Send message to every active user
<< -msg        Send private message
<< -list       Show all clients and the status
<< -arq        Send private archive file
<< -help       Show help
> Escolha sua opção: █
```

Figura 5 – Menu de funcionalidades para o usuário recém conectado.

Para o envio de mensagens, é necessário o cliente utilizar o comando msg, onde será solicitado o destino da mensagem, além da mensagem que deseja ser enviada.

Após a confirmação da mensagem, o servidor irá fazer uma busca entre os clientes que estão online para poder encaminhar o pacote de maneira correta.

O destinatário irá receber a mensagem com a indicação de quem a enviou, permitindo assim identificar de onde veio a mensagem e, caso queira, responder para o cliente correto.



```
barai@DESKTOP-TKFK37R:/mnt/c/Users/carlo/Documents/Faculdade 20
21/RedesA/Projeto1$ ./cl
Select your nickname: teste2
Connect to Server: 127.0.0.1:1234
You: 127.0.0.1:54391
<< -quit      Quit chatroom
<< -all        Send message to every active user
<< -msg        Send private message
<< -list       Show all clients and the status
<< -arq        Send private archive file
<< -help       Show help
> Escolha sua opção: msg
> Para quem deseja mandar a mensagem: teste2
> ola amigo
<< -quit      Quit chatroom
<< -all        Send message to every active user
<< -msg        Send private message
<< -list       Show all clients and the status
<< -arq        Send private archive file
<< -help       Show help
> Escolha sua opção: █
```

Figura 6 - Mensagem enviada de um cliente para outro.

Caso seja de interesse do usuário enviar um arquivo, o processo é semelhante com o de enviar mensagens, porem ao invés de ser escrita uma mensagem, um arquivo é aberto e seu conteúdo é enviado ao destinatário escolhido. Já o servidor, por sua vez, faz os mesmos passos de quando é enviada uma mensagem.


```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
service: -all
Msg for all clients
Send to sockfd 4: "teste1: ola pessoal"
Send to sockfd 6: "teste1: ola pessoal"
no whilemsg for all clients
teste1 (5) leave the chatroom.
Send to sockfd 4: "teste1(127.0.0.1) leave the chatroom."
Send to sockfd 6: "teste1(127.0.0.1) leave the chatroom."
no whileClient 127.0.0.1:54589 connected.
teste4(127.0.0.1)(5) join the chatroom.
no whilerecebi o serviço
service: -all
Msg for all clients
Send to sockfd 4: "teste4: ola pessoal"
Send to sockfd 6: "teste4: ola pessoal"
no while]

Select your nickname: teste4
Connect to Server: 127.0.0.1:1234
You: 127.0.0.1:54589
<< -quit Quit chatroom
<< -all Send message to every active user
<< -msg Send private message
<< -list Show all clients and the status
<< -arq Send private archive file
<< -help Show help
> Escolha sua opção: -all
> ola pessoal
<< -quit Quit chatroom
<< -all Send message to every active user
<< -msg Send private message
<< -list Show all clients and the status
<< -arq Send private archive file
<< -help Show help
> Escolha sua opção: ]

barai@DESKTOP-TKFK37R:/mnt/c/Users/carla/Documents/Faculdade 2021/RedesA/Projeto1$ ./cli
Select your nickname: teste3
Connect to Server: 127.0.0.1:1234
You: 127.0.0.1:54571
<< -quit Quit chatroom
<< -all Send message to every active user
<< -msg Send private message
<< -list Show all clients and the status
<< -arq Send private archive file
<< -help Show help
> Escolha sua opção:
teste1: ola pessoal
>
teste1(127.0.0.1) leave the chatroom.
>
teste4: ola pessoal
> ]

Select your nickname: teste2
Connect to Server: 127.0.0.1:1234
You: 127.0.0.1:54389
<< -quit Quit chatroom
<< -all Send message to every active user
<< -msg Send private message
<< -list Show all clients and the status
<< -arq Send private archive file
<< -help Show help
> Escolha sua opção:
teste1: ola amigo
>
teste1: ola pessoal
>
teste1(127.0.0.1) leave the chatroom.
>
teste4: ola pessoal
> ]
```

Figura 7 - Mensagem enviada ao arquivo

5. Análise dos Resultados

Analizando os testes realizados no experimento, pudemos notar que o protocolo TCP é muito eficaz no tratamento de envio e recebimento de mensagens ou arquivos, seja para 1 ou mais destinatários, isso graças a confiabilidade no recebimento e confirmação dos pacotes de dados, uma vez que estes são assegurados através de métodos dentro da construção do protocolo. Apesar de não ser muito aconselhado em usar com muitos dados, devido a sua latência pela alta na demanda dos dados, ele atendeu todas as expectativas e realizou as tarefas de modo rápido e prático.

6. Conclusão

O projeto permitiu a consolidação do aprendizado adquirido nas aulas nos experimentos anteriores, reforçando o conhecimento sobre protocolos de redes, em especial o TCP, e comunicação entre clientes via servidor, o que permitiu a todos os envolvidos ter a capacidade de desenvolver um serviço de comunicação entre servidor e cliente caso necessário.

O grupo não sofreu muitas dificuldades ao longo do processo de produção do projeto, uma vez os problemas apresentados foram corrigidos após reuniões e debates sobre as possíveis soluções. O projeto foi muito importante para colocar em prática o conteúdo passado em sala e para amadurecer sobre o uso de processos de rede.

7. Link do vídeo

<https://youtu.be/wH2EKX6JsNg>